

Anwendung: Das Travelling Salesman Problem

Matthias Raab

1. Einführung	2
2. Kodierungen und zugehörige Operatoren	2
2.1 Adjacency Representation	2
2.1.1 Alternating Edges Crossover	3
2.1.2 Subtour Chunks Crossover	3
2.1.3 Heuristic Crossover	3
2.2 Ordinal Representation	3
2.3 Path Representation	4
2.3.1 PMX	4
2.3.2 OX	5
2.3.3 CX	6
2.3.4 ERX	6
2.4.5 Mutation	7
2.4 Matrixdarstellungen	8
2.4.1 Durchschnitt	8
2.4.2 Vereinigung	9
3. Bewertungsfunktion	10
4. Evolutionsstrategien als Lösungsansatz	10
5. Lokale Optimierung	10
6. Konkrete Versuchsergebnisse	11
7. Quellen	13

1. Einführung: Das Travelling Salesman Problem

Das Problem des Handelsreisenden ist ein sehr bekanntes Problem der Graphentheorie. Das Ziel besteht darin, eine Route zwischen n Städten zu finden, die jede Stadt genau einmal besucht und dabei eine gewisse Länge nicht überschreitet.

Formale Definition:

Geg: $n \times n$ Matrix (M_{ij}) , die die Entfernungen zwischen den „Städten“ enthält und eine maximale „Rundreiselänge“ k .

Frage: Gibt es eine Permutation π , dass
$$\sum_{i=1}^{n-1} M_{\pi(i), \pi(i+1)} + M_{\pi(n), \pi(1)} \leq k$$

Heute wird das TSP häufig als reines Optimierungsproblem betrachtet, d.h. es wird nicht mehr eine gewisse Länge vorgegeben, die nicht überschritten werden soll, sondern es wird nach der kürzesten Route gesucht (meistens ist es ja auch das, was man erreichen will).

Das TSP ist NP-vollständig und somit insbesondere NP-hart. Demnach ist es extrem unwahrscheinlich, effiziente Algorithmen zur Lösung dieses Problems zu finden (es sei denn $P=NP$). Der „primitive“ Algorithmus, der einfach alle Möglichkeiten durchprobiert hat eine Komplexität von $o(n!)$. Es muss zwar aufgrund der NP-Vollständigkeit möglich sein, eine Komplexität von $2^{O(n)}$ zu erreichen (und es ist auch möglich), aber selbst dies ist aufgrund der Größenordnungen, in denen es häufig vorkommt, nicht tragbar.

Zudem ist oftmals eine Lösung, die nur um wenige Prozent von der optimalen Lösung abweicht, völlig ausreichend und falls diese Lösung in relativ kurzer Zeit mit günstiger Hardware zu erzielen ist, dann ist dies mit Sicherheit optimaler als eine perfekte Lösung die mit extremem Zeitaufwand auf teuren Supercomputern ermittelt wird. Und ab einer gewissen Problemgröße ist es in realer Zeit sowieso nicht mehr möglich eine optimale Lösung zu ermitteln.

2. Repräsentation/Kodierung

Es gibt sicherlich viele Möglichkeiten eine mögliche Tour zu repräsentieren, wobei allerdings die klassische Form des Binärstrings für dieses Problem kaum geeignet ist, da sich eigentlich bei jeder Operation (sei es Crossover oder Mutation) ungültige Lösungen ergeben und somit aufwendige Reparaturmaßnahmen notwendig werden. Allgemein sind Binärdarstellungen für Permutationen nur sehr bedingt geeignet (s. Vortrag Frank Förster).

Im folgende werden einige mögliche Darstellungen vorgestellt.

2.1 Adjacency Representation

Hier bei wird eine Tour der Länge n durch eine Liste von natürlichen Zahlen dargestellt. Die Zahl (Stadt) j ist an i -ter Stelle aufgelistet, falls die Tour von i nach j führt.

Beispiel für $n = 7$:

(6 1 5 3 2 7 4) repräsentiert die Tour 1-6-7-4-3-5-2

Jede Tour hat somit genau eine Adjazenzdarstellung. Es ist jedoch möglich, dass eine solche Darstellung eine ungültige Tour darstellt: z.B. (3 4 1 5 6 7 2) führt zu zwei geschlossenen Kreisen, nämlich 1-3-1 und 2-4-5-6-7-2.

Der klassische Crossover ist in dieser Darstellung nicht praktikabel, da er ungültige Lösungen erzeugt (z.B können Zahlen danach mehrfach in der Liste vorkommen). Es wurden aber speziell für diese Darstellung Crossover-Operatoren definiert, die diese Probleme umgehen. Für die Mutation werden in [1] keine Operatoren erwähnt, aber die meisten für die Path Representation (siehe Abschnitt 2.3) können mit nachfolgender Überprüfung auf Gültigkeit verwendet werden.

2.1.1 Alternating Edges Crossover

Hierbei wird ein Nachfahre erzeugt, indem abwechselnd (und auch zufällig) Kanten vom ersten und zweiten Elternteil gewählt werden. Sollte eine dieser Kanten vorzeitig ein Kreis erzeugen, so wird eine zufällige Kanten aus der Menge der noch übrigen Kanten gewählt.

Beispiel für $n = 7$:

$p_1 = (3\ 1\ 7\ 2\ 4\ 5\ 6)$ und $p_2 = (4\ 3\ 7\ 6\ 2\ 5\ 1)$ könnten folgenden Nachkommen erzeugen:

$o = (3\ 1\ 7\ 6\ 2\ 5\ 4)$

Und zwar geschieht dies, indem vom ersten Elternteil der erste Knoten (die 3) gewählt, dann vom ersten Elternteil die 1, dann die 7, 6, 2 und die 5 vom zweiten. Nun kann für die letzte Position weder die 6 vom ersten noch die 1 vom zweiten Elternteil eingefügt werden, ohne den Kreis frühzeitig zu schließen. Es wird also die 4 gewählt, da sie noch übrig ist.

2.1.2 Subtour Chunks Crossover

Dieser Operator wählt abwechselnd Teiltouren (mit zufälliger Länge) von beiden Elternteilen und kombiniert diese. Sollte es hier wieder zu ein frühzeitigen Kreisschließung kommen, so wird die entsprechende Kante durch eine zufällige der noch nicht verwendeten Kanten ersetzt, die keinen Kreis erzeugt.

2.1.3 Heuristic Crossover

Hier wird eine zufällige Stadt als Startpunkt für den Nachkommen gewählt. Daraufhin werden beide Kanten von beiden Elternteilen, die aus dieser Stadt führen, verglichen und die kürzere gewählt. Mit der so neu „erreichten“ Stadt wird dieser Vorgang fortgesetzt. Sollte die Kante aber ein frühzeitigen Kreis erzeugen, so wird stattdessen eine zufällige noch unbenutzte Kante gewählt, bei der dies nicht der Fall ist.

Man kann bei vorzeitiger Kreisbildung alternativ auch statt einer zufälligen eine längere der in den Elternteilen angrenzen Kanten verwenden, falls diese keinen Kreis erzeugt. Zusätzlich kann man die zufällige Kantenwahl dahingehend verändern, dass stets die kürzeste Kante, die zu keinem Kreis führt, verwendet wird.

Laut [1] ergeben sich die besten Ergebnisse in der Adjazenzdarstellung bei Verwendung des heuristic crossover, wobei deren Qualität allerdings zu wünschen übrig lässt. Die Ergebnis für 50, 100 und 200 Städte lagen zwischen 16% und 27% überhalb des Optimums, und das bei ca. 15000, 2000 und 25000 Generationen!

2.2 Ordinal Representation

Auch in dieser Darstellung ist eine Tour eine Liste von n Städten, wobei das i -te Element der Liste zwischen 1 und $n - i + 1$ liegen muss. Diese Liste repräsentiert eine Tour unter Zunahme einer sortierten Referenzliste C (z.B. $C = (1\ 2\ 3\ 4\ 5\ 6\ 7)$) in folgender Art und Weise:

$(2\ 3\ 1\ 4\ 1\ 2\ 1)$ repräsentiert die Tour 2-4-1-7-3-6-5.

Die 2 am Anfang besagt: Nehme das zweite Element in C als ersten Wegpunkt und lösche es aus C .

Die nächsten Zahlen in der Darstellung werden auf die selbe Art und Weise interpretiert. Es ist klar, dass z.B. die letzte Zahl immer 1 sein muss, da C zu diesem Zeitpunkt nur noch 1 Element enthält. Auch klar ersichtlich ist, dass in dieser Darstellung keine ungültigen Routen vorkommen können. Der Hauptvorteil der Ordinaldarstellung ist allerdings, dass der klassische Crossover funktioniert.

Beispiel für $n=7$:

$p_1 = (2\ 3\ 1\ | \ 4\ 1\ 2\ 1)$ und

$p_2 = (6\ 4\ 5\ | \ 2\ 3\ 1\ 1)$ ergeben mit dem Crossover Punkt „|“ folgende Nachkommen:

$o_1 = (2\ 3\ 1\ 2\ 3\ 1\ 1)$ und

$o_2 = (6\ 4\ 5\ 4\ 1\ 2\ 1)$

was den Touren 2-4-1-5-7-3-6 und 6-4-7-5-1-3-2 entspricht.

Bei einem solchen Crossover ist es jedoch so, dass die Teil-Tour linksseitig des Crossover Punktes nicht verändert wird, während die rechte Seite in ziemlich zufälliger Weise verändert wird. Mutation ist möglich, indem z.B. einzelne Werte im Rahmen des für ihre Position gültigen Wertebereichs verändert werden.

Da diese Darstellung in Kombination mit dem klassischen Crossover Operator aber relativ schlechte Ergebnisse liefert [1], ist auch sie nicht der optimale Weg zur Lösung des TSP.

2.3 Path Representation

Die einfachste Art eine Tour darzustellen ist die Reihenfolge der Elemente in einer Liste direkt als Weg zu interpretieren. (3 4 7 6 1 2 5) ist also hier die Darstellung der Tour 3-4-7-6-1-2-5.

2.3.1 Partially Matched Crossover (PMX)

Es wird eine Teiltour aus einem Elternteil übernommen und versucht die Reihenfolge und Position möglichst vieler Städte aus dem anderen Elternteil zu erhalten. Dies geschieht, indem zwei zufällige Crossoverpunkte gewählt werden, zwischen denen dann die Werte vertauscht werden. Dann werden die Städte links und rechts der Crossoverpunkte wieder an ihrer Originalposition eingefügt, sofern sie die Gültigkeit der Darstellung nicht verletzen. Sollte dies aber der Fall sein, wird stattdessen der Wert eingefügt der vor der Crossover Operation an der Stelle stand, an der nun der gleiche Wert steht wie der, der eingetragen werden soll.

Beispiel für $n = 7$:

$p_1 = (1\ 4\ | \ 6\ 7\ 5\ 3\ | \ 2)$ und

$p_2 = (5\ 2\ | \ 7\ 6\ 4\ 3\ | \ 1)$

erzeugen durch Vertauschung zuerst folgende Nachkommen

$o_1 = (x\ x\ | \ 7\ 6\ 4\ 3\ | \ x)$

$o_2 = (x\ x\ | \ 6\ 7\ 5\ 3\ | \ x)$

folgende Werte können ohne Konflikt an ihre Originalpositionen:

$o_1 = (1\ x\ | \ 7\ 6\ 4\ 3\ | \ 2)$

$o_2 = (x\ 2\ | \ 6\ 7\ 5\ 3\ | \ 1)$

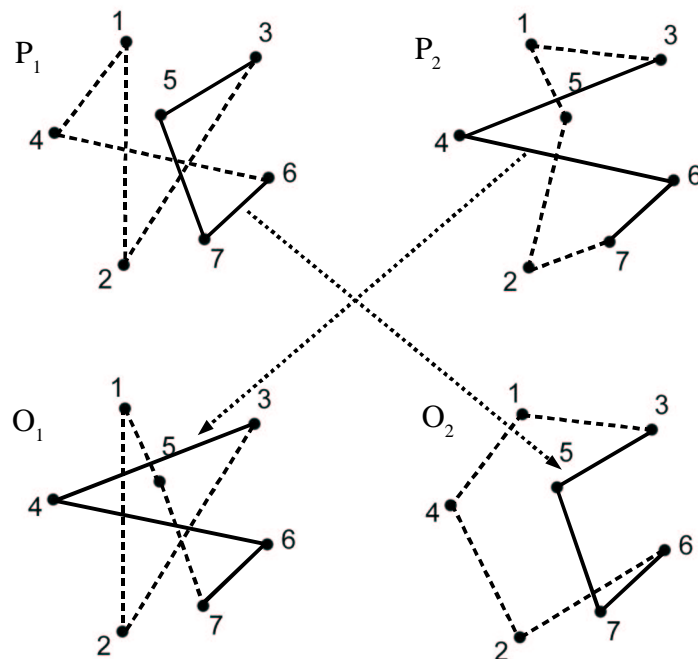
In o_1 konnte die 4 nicht eingefügt werden, da sie schon an Stelle 5 steht. Also wird sie durch das Element, welches zuvor an Stelle 5 stand, ersetzt (die 5). Analog wird bei o_2 verfahren. Es ergeben sich also folgende Nachkommen:

$o_1 = (1\ 5\ 7\ 6\ 4\ 3\ 2)$

$o_2 = (4\ 2\ 6\ 7\ 5\ 3\ 1)$

Beim PMX werden Ähnlichkeiten hauptsächlich in absoluter Position erhalten aber auch Ähnlichkeiten in der Reihenfolge (natürlich innerhalb der Crossoverpunkte).

Graphisch sind das Ganze in obigen Beispiel wie folgt aus:



2.3.2 Order Crossover OX

Prinzipiell arbeitet OX ähnlich wie PMX, es gibt aber signifikante Änderungen bei der Erhaltung der Gültigkeit. Die Elternteile werden wieder durch je zwei Crossoverpunkte unterteilt, aber hier nicht vertauscht. Die Lücken links und rechts (zuerst rechts) werden dann in der Folge gefüllt, in der die Werte im anderen Elternteil startend vom rechten Crossoverpunkt vorkommen. Werte, die schon zwischen den Crossoverpunkten vorkommen, werden hierbei übersprungen.

Beispiel für $n = 7$:

$p_1 = (1\ 4\ |6\ 7\ 5\ 3|\ 2)$ und

$p_2 = (5\ 2\ |7\ 6\ 4\ 3|\ 1)$

beim OX vorläufig folgende Nachkommen

$o_1 = (x\ x\ |6\ 7\ 5\ 3|\ x)$

$o_2 = (x\ x\ |7\ 6\ 4\ 3|\ x)$

Dann werden in o_1 die Werte in der Reihenfolge, in der sie (nach dem zweiten Crossoverpunkt) in o_2 vorkommen, hinter dem zweiten Crossoverpunkt eingefügt (5 2 7 6 4 3 1). Die Werte 6 7 5 3 werden hierbei übersprungen, da sie zwischen den Crossoverpunkten schon vorhanden sind (\Rightarrow 5 2 4). Bei o_2 wird analog verfahren. Die Nachkommen sind also:

$o_1 = (2\ 4\ 6\ 7\ 5\ 3\ 1)$

$o_2 = (5\ 2\ 7\ 6\ 4\ 3\ 1)$

Es ist ersichtlich, dass beim OX teilweise auch die absoluten Positionen in der Liste (zwischen den Crossoverpunkten) erhalten bleiben, ansonsten aber im Vergleich zum PMX eher auf die relative

Reihenfolge geachtet wird. Die Veränderung der absoluten Position kann ein Problem sein, da die Position eine wichtige Eigenschaft der Pfad Repräsentation ist (in Bezug auf ihre Eindeutigkeit). Die Touren (1 3 6 5 4 7 2) und (7 2 1 3 6 5 4) sind z.B. identisch.

2.3.3 Cycle Crossover CX

Beim Cycle Crossover wird der Wert an einer zufälligen Position in einem Elternteil an die selbe Stelle im Nachkommen übernommen. Dann wird ausgehend von der Stelle, an der dieser Wert im anderen Elternteil steht, dieser Vorgang so lange wiederholt, bis sich ein Kreis bildet (man an einem Wert ankommt, den man schon übernommen hat). Die verbleibenden Stellen werden dann in der Reihenfolge aufgefüllt, in der die übrigen Werte im anderen Elternteil vorkommen.

Beispiel für $n = 7$, Startposition 2:

$$p_1 = (1\ 4\ 6\ 7\ 5\ 3\ 2)$$

$$p_2 = (5\ 2\ 7\ 6\ 4\ 3\ 1)$$

Als erstes wird die 4 aus p_1 an Stelle 2 in o_1 übernommen.

$$o_1 = (x\ 4\ x\ x\ x\ x\ x)$$

Die 4 steht in p_2 an Position 5, also wird nun Position 5 von p_1 (dort steht die 5) an die selbe Position in o_1 übernommen

$$o_1 = (x\ 4\ x\ x\ 5\ x\ x)$$

Die 5 steht in p_2 an Position 1, also wird nun Position 1 von p_1 (dort steht die 1) an die selbe Position in o_1 übernommen.

$$o_1 = (1\ 4\ x\ x\ 5\ x\ x)$$

Die 1 steht in p_2 an Position 7, also wird nun Position 7 von p_1 (dort steht die 2) an die selbe Position in o_1 übernommen.

$$o_1 = (1\ 4\ x\ x\ 5\ x\ 2)$$

Die 2 steht in p_2 an Position 2, somit würde jetzt also Position 2 (die 4) an Position 2 in o_1 übernommen werden, was jedoch schon geschehen ist. Der Kreis ist nun geschlossen (Position 2 war die Startposition).

Analog ergibt sich durch Ausführung bis zur Kreisbildung o_2 :

$$o_2 = (5\ 2\ x\ x\ 4\ x\ 1)$$

Nun werden die unbelegten Werte mit den noch übrigen aus dem anderen Elternteil ersetzt (bei o_1 jetzt also 7, 6, 3; bei o_2 6, 7, 3):

$$o_1 = (1\ 4\ 7\ 6\ 5\ 3\ 2)$$

$$o_2 = (5\ 2\ 6\ 7\ 4\ 3\ 1)$$

2.3.4 Edge Recombination Crossover ERX

Der ERX ist als rein kantenbasierter Operator konzipiert, unter der Betrachtung, dass eigentlich die Kanten die wichtigen Informationen im TSP (nämlich die Distanz) enthalten. Hier ist die Reihenfolge der Städte irrelevant, nur die Verbindungen zwischen ihnen zählen. Es wird also versucht, einen Nachfahren aus den Kanten, die in beiden Eltern vorkommen, zu erzeugen. Konkret geschieht dies, indem eine Kantenliste erzeugt wird, in der für jede Stadt die von ihr ausgehenden Kanten (aus beiden Eltern) zu anderen Städten aufgeführt sind.

Beispiel für $n = 7$:

Die Eltern $p_1 = (1\ 4\ 6\ 7\ 5\ 3\ 2)$ und $p_2 = (5\ 2\ 7\ 6\ 4\ 3\ 1)$ ergeben folgende Kantenliste:

Stadt	Kanten zu anderen Städten
1	4,2,3,5
2	3,1,5,7
3	5,2,4,1
4	1,6,3
5	7,3,1,2
6	4,7
7	6,5,2

Der Nachkomme wird nun erzeugt, indem zuerst als Startpunkt eine beliebige Stadt eines Elternteil genommen wird. Nun wird von den Städten, zu denen diese Anfangsstadt eine Verbindung hat, die mit den wenigsten Kanten als nächster Punkt der Tour gewählt (sollten es mehrere sein, dann zufällige Auswahl). Dies wird solange wiederholt, bis die Tour komplett ist.

Wählen wir also in unserem Fall z.B. Stadt 1 als Ausgangspunkt. Stadt 1 hat Kanten zu den Städten 4, 2, 3 und 5. Wir wählen also Stadt 4 als neuen Wegpunkt, da sie am wenigsten weiterführende Kanten in der Liste hat. Hier bleiben nur Stadt 6 und 3 zur Auswahl, da Stadt 1 schon besucht wurde. Da Stadt 6 weniger Kanten hat als Stadt 3, wird sie als nächster Wegpunkt gewählt.

Bisher ergibt dies folgende Tour (1 4 6 x x x).

Führt man dies fort (mit entsprechender zufälliger Auswahl), ergibt sich folgender Nachkomme:
 $o_1 = (1\ 4\ 6\ 7\ 5\ 3\ 2)$

Man sieht, dass o_1 nur aus Kanten beider Eltern besteht. Natürlich ist es möglich, in eine Stadt ohne vorhandene weiterführende Kante zu kommen, aber dieser *edge failure* genannte Vorfall tritt bei der Wahl der Stadt mit den wenigsten weiterführenden Kanten nur sehr selten auf (1%-1,5% aller Fälle [1]). Die Wahrscheinlichkeit ist dann am höchsten, wenn die Eltern sehr ähnlich sind und somit die Kantenliste wenige Einträge hat (Extremfall: $p_1 = p_2 \Rightarrow 2n$ Einträge in der Kantenliste)

In der Path Representation führt der ERX zu den besten Ergebnissen, gefolgt von OX, PMX und CX ([1], [2]). Dies ist darauf zurückzuführen, dass er ein rein kantenbasierter Operator ist und, da in die Bewertungsfunktion nur Kanteninformationen einfließen, somit die wichtigsten Informationen verarbeitet.

2.3.5 Mutation

Für die Path Representation wurden in [1] im Gegensatz zu den anderen vektorbasierten Darstellungen auch Mutationsoperatoren angegeben:

- Inversion
Innerhalb zweier Schnittpunkte wird die eingeschlossene Teiltour umgedreht
z.B. (1 3 |4 5 7 6| 2) \rightarrow (1 3 |6 7 5 4| 2). Dies kann zum Beispiel Kreuzungen in Touren entfernen.
- Insertion
Eine Stadt wird an ihrer ursprünglichen Position entfernt und an anderer zufälliger Position wieder eingesetzt.
- Displacement
Eine Teiltour wird aus der Gesamttour entfernt und an zufälliger Position wieder eingesetzt.
- Reciprocal Exchange
Zwei Städte werden vertauscht.

Mutative Schrittweitenregelung

Um die Wirkung der Mutation bei Annäherung an das Optimum abzuschwächen, kann eine Mutative Schrittweitenregelung verwendet werden, z.B bei *inversion* eine Beschränkung der Länge der umzudrehenden Teiltour. Das einzige Problem ist hierbei jedoch ein Abschätzung der

minimalen Tour. Die kann aber bei zufällig verteilten Städten durch folgende empirische Formel geschehen:

$$L = k \cdot \sqrt{n \cdot R} \quad \text{mit:}$$

$n = \# \text{ Städte}$, $R = \text{Fläche des Quadrates, das die Städte umschließt}$ und $k \approx 0,765$.

2.4 Matrixdarstellungen

Wie die gute Performance des ERX zeigt, sind weniger die absoluten Positionen, sondern vielmehr die Kanteninformationen, das, was eine gute Tour ausmacht. Die Path Representation ist somit also mit Sicherheit nicht die perfekte Darstellung für das TSP. Deshalb wurden andere, nicht vektorbasierte, Kodierungen entwickelt.

Die von Fox und McMahon eingeführte Darstellung einer Tour ist eine binäre Matrix $M = (m_{ij})$, wobei m_{ij} nur dann 1 ist, falls die Stadt i in der Tour vor der Stadt j besucht wird.

z.B. 2-3-1-5-6-4-7 wird durch folgende Matrix repräsentiert: $T = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Diese $n \times n$ Matrizen haben folgende Eigenschaften:

1. Die Anzahl Einsen ist $\frac{n(n-1)}{2}$
2. $m_{ii} = 0$ für alle i
3. $m_{ij} = 1$ und $m_{jk} = 1 \Rightarrow m_{ik} = 1$ (Transitivität)

Sollten Bedingung 2. und 3. erfüllt sein und nur die Anzahl der Einsen zu gering sein, ist es möglich die Matrix zu „reparieren“, um eine gültige Tour zu erzeugen. Durch diese Betrachtung ist es möglich Crossover Operationen auf Matrizen zu definieren.

Mutation wurde jedoch von Fox und McMahon keine definiert.

2.4.1 Durchschnitt

Da bei der Durchschnitt zweier wie oben codierter Matrizen, die Anzahl der Einsen $\frac{n(n-1)}{2}$ nicht übersteigt und sowohl Bedingung 2. als auch 3. eingehalten werden, ist die Erzeugung einer legalen Tour möglich. Der entstehende Nachfahre muss allerdings noch zu einer kompletten Tour erweitert werden.

Beispiel für $n = 7$:

Die Touren 1-3-6-5-2-7-4 und 4-1-2-3-7-6-5 haben folgende Matrixdarstellungen:

$$P_1 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad \text{und} \quad P_2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Der Durchschnitt beider Matrizen ist:

$$O_{\text{vorläufig}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Die Teilordnung, die durch O vorgegeben wird, ist:

- Stadt 1 kommt vor Stadt 2, 3, 5, 6 und 7.
- Stadt 2 kommt vor Stadt 7.
- Stadt 3 kommt vor Stadt 5, 6, 7.
- Stadt 6 kommt vor Stadt 5.

Nun muss O noch zu einer gültigen Tour erweitert werden. Dies geschieht, indem ein Elternteil ausgewählt wird und einige Einsen aus ihm übernommen werden. Vervollständigt wird die Matrix durch eine Analyse der Zeilen- und Spaltensumme.

2.4.2 Vereinigung

Die Vereinigung zweier disjunkter Teilmengen (eine von jedem Elternteil) kann durchgeführt werden, ohne dass eine der Voraussetzungen der Gültigkeit verletzt wird. Dazu wird die Menge der Städte in zwei zusammenhängende Gruppen geteilt, etwa $\{1, 2, 3, 4\}$ und $\{5, 6, 7\}$. Nun werden die die Bits in der ersten Gruppe vom ersten Elternteil und die Bits von der zweiten Gruppe vom zweiten Elternteil an der entsprechenden Stellen in den Nachkommen kopiert. Für obiges Beispiel sieht dies dann wie folgt aus:

$$o_{\text{vorläufig}} = \begin{pmatrix} 0 & 1 & 1 & 1 & x & x & x \\ 0 & 0 & 0 & 1 & x & x & x \\ 0 & 1 & 0 & 1 & x & x & x \\ 0 & 0 & 0 & 0 & x & x & x \\ x & x & x & x & 0 & 0 & 0 \\ x & x & x & x & 1 & 0 & 0 \\ x & x & x & x & 1 & 1 & 1 \end{pmatrix}$$

Diese Matrix wird dann wiederum durch eine Analyse der Zeilen- und Spaltensumme vervollständigt.

In Versuchen mit verschiedenen Stadtanordnungen zeigt sich eine wesentliche Eigenschaft dieser Matrixdarstellung in Kombination mit den eben besprochenen Crossover Operatoren. Der Algorithmus macht nämlich auch dann Fortschritte, wenn kein Elitismus verwendet wird (die Besten nicht gesichert werden), was weder bei ER noch bei PMX der Fall ist ([1]).

3. Bewertungs- / Fitnessfunktion

Beim TSP bietet sich klarerweise die Länge der durch ein Individuum erzeugten Tour als Bewertungsfunktion zu verwenden, also:

$$\phi(\pi) = \sum_{i=1}^{n-1} M_{\pi(i), \pi(i+1)} + M_{\pi(n), \pi(1)}$$

wobei (M_{ij}) die entsprechende Distanzmatrix ist, und π die die Permutation, die das entsprechende Individuum darstellt.

Anmerkung:

Es ist klar, dass $M_{ii} = 0$ für alle i gelten muss. Desweiteren ist diese im Allgemeinen auch symmetrisch (d.h. die Distanz zwischen Stadt i und Stadt j ist dieselbe wie zwischen Stadt j und Stadt i). Sollte die Bewertung einer Kante aber „richtungsabhängig“ sein (man kann ja annehmen, dass zwischen beiden Städten eine „Steigung“ existiert, welche die eine Richtung erschwerlicher macht als die andere), muss dies nicht der Fall sein. Dies entspricht aber nicht der Definition des klassischen TSP (es ist aber unter dem Namen Asymmetrisches TSP bekannt) und es wird in der Literatur auch nicht darauf eingegangen, inwiefern dies den Wirkungsgrad der einzelnen Crossover-Operatoren in den verschiedenen Darstellungen beeinflusst. Klar ist, dass in diesem Fall z.B. der Mutations-Operator *inversion* erheblich gravierendere Änderungen bewirkt als normal.

4. Evolutionsstrategien als Lösungsansatz

Es gibt Ansätze, das TSP über ES zu optimieren, wobei hauptsächlich auf die bei der Path Representation erwähnten Mutations-Operatoren zurückgegriffen wird.

Weitere Möglichkeit:

Ein Vektor mit n Gleitkommazahlen (n ist die Anzahl der Städte) wird wie bei jedem kontinuierlichen Problem, das über ES gelöst wird verwendet. Eine Tour wird hierbei durch die Position der Zahlen repräsentiert, wobei die k -größte Zahl die k -te Stadt repräsentiert und ihre Position im Vektor ihrer Position in der Tour entspricht.

Bsp:

(3.14, 2.3, -1.0, 2.34, -2.33, 0.21, -0.99) repräsentiert die Tour 5-3-7-6-2-4-1, da die kleinste Zahl (-2.33) an Position 5 steht, die zweitkleinste an 3, usw.

5. Lokale Optimierung

Da die Suche nach dem absoluten Optimum aufgrund der NP-Härte des TSP wohl nie in effizienter Weise realisiert werden kann und man sich daher eher auf die Approximation dieses Optimums konzentriert, folgt nun eine Betrachtung, wie man dies mit effizienten Algorithmen lösen kann, die sich auf lokale Optima konzentrieren. Die Idee dahinter ist, für eine gegebene Tour eine Menge benachbarter Touren zu erzeugen (z.B. über 2-opt, wobei nur 2 Kanten verändert werden) und sie

dann durch eine bessere Tour aus diesen Nachbarn zu ersetzen. Diese lokale Optimierungen (z.B. 2-opt) sind effizient mit GAs zu realisieren.

Einer Genetischer Algorithmus für lokale Suche kann wie folgt aussehen:

- (i) Über lokale Optimierung wird jede Tour der Population (Größe μ) durch eine lokal optimale ersetzt.
- (ii) Die Population wird um λ Touren, die durch Rekombination einiger Touren entstanden sind ergänzt,
- (iii) Die ergänzten Touren werden analog (i) durch lokale Optima ersetzt.
- (iv) Die Population wird wieder auf die Ursprungsgröße μ reduziert (nach entsprechenden Selektionsparametern).
- (v) Zurück zu Schritt (ii), Wiederholung bis Abbruchkriterium erreicht.

Man sieht hier eine deutliche Ähnlichkeit zu $(\mu + \lambda)$ -ES.

Die Testergebnisse solcher auf lokaler Optimierung aufgebauter Algorithmen sind teilweise ziemlich gut. Bei 532 Städten wurde ein Tour gefunden, die nur um 0.06 % vom Optimum abweicht[1].

6. Konkrete Versuchsergebnisse

Nun folgen noch einige Ergebnisse meiner Implementierung eines GA zur Optimierung des TSP. Verwendet habe ich die Path Representation, als Crossover-Operatoren kommen wahlweise PMX und ERX zum Einsatz, als Mutationart wurde *inversion* verwendet. Es kam eine Wettkampfselektion zum Einsatz und die Besten einer Generation wurden beibehalten (Elitismus).

Grundlage war ein Problem mit 26 Städten, dessen optimale Lösung mir bekannt war (Fricker 26, Lösung: 937 [3]). Die Anzahl der Möglichkeiten an Touren ist $(n - 1)! / 2$, also hier ungefähr $8 \cdot 10^{24}$.

Das beste durch den Algorithmus erzielte Ergebnis war 971 unter der Verwendung von ERX bei einer Populationsgröße von 500, einer Mutationswahrscheinlichkeit von 0,2 und einer Crossoverwahrscheinlichkeit von 0,9. Das Ergebnis wurde in Generation 87 erreicht, die Laufzeit war unter 10 Sekunden.

Beobachtungen:

Allgemeines

- Die fittesten Individuen (Touren) der Ausgangspopulation waren in etwa doppelt so lang wie das Optimum. Selbst bei großen Populationen (50 000 +) waren sie noch etwa 1,7 mal so lang.
- Eine hohe Mutationswahrscheinlichkeit ($\geq 0,2$) verhindert eine frühzeitige Konvergenz.
- Mutation ohne hohe Crossoverwahrscheinlichkeit liefert keine bzw. nur sehr langsam brauchbare Ergebnisse.
- Die Performance von ERX war (in den ersten Generationen) nicht so gut wie in [2]. Dort führt ERX von Anfang an zu bessern Ergebnissen wie PMX.

PMX

- Bei kleiner Population schnelle Konvergenz (typischerweise zwischen 10 und 20 Generationen). Eine hohe Generationenanzahl liefert kaum neue Ergebnisse.
- Brauchbare Ergebnisse nur bei hoher Population (über 400), sonst Konvergenz weit ab vom Optimum.

ERX

- Bei kleiner Population langsame Konvergenz (über 30 Generationen).
- Nicht ganz so hohe Population wie bei PMX nötig.
- Beste Ergebnisse bei hoher Generationenanzahl.

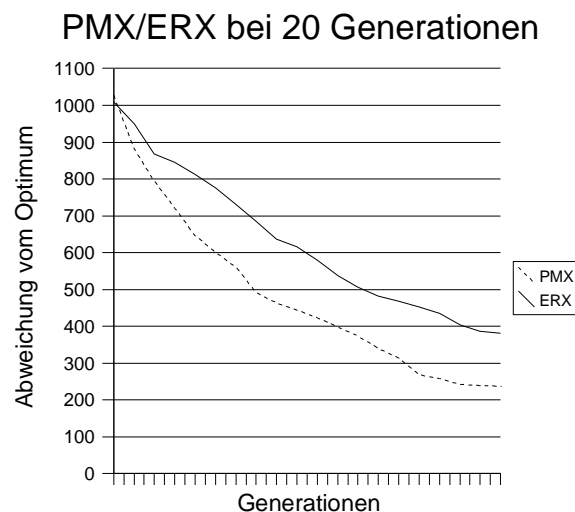
Insgesamt scheint PMX schneller zu brauchbaren Ergebnissen zu führen, wohingegen ERX insgesamt bessere Ergebnisse liefert.

Die Laufzeit war hauptsächlich abhängig von der Größe der Population und stieg bei großen Populationen in späteren Generationen auch an (hohe Ähnlichkeit der Individuen verlangsamt die Sortierung, die in meinem Algorithmus in jeder Generation durchgeführt wird). Bei kleinerer Population fällt dieser Effekt gering aus, und es ist somit möglich sehr viele Generationen zu durchlaufen, was insbesondere bei Verwendung des ERX gute Ergebnisse in akzeptabler Zeit liefert.

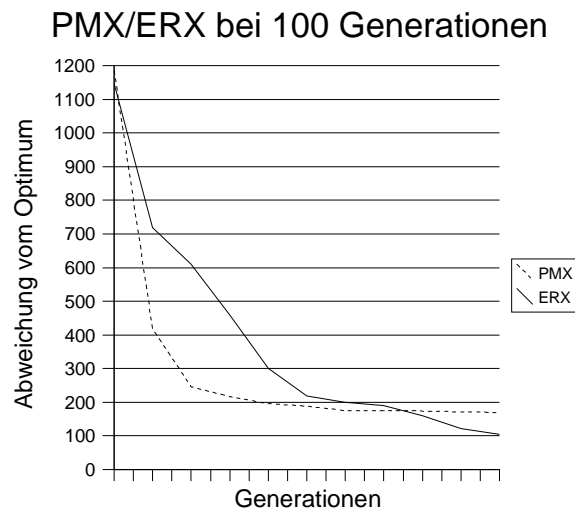
Im folgenden wurde eine Mutationswahrscheinlichkeit von 0,2 und eine Crossoverwahrscheinlichkeit von 0,9 verwendet. Im ersten Fall wurde über drei Versuchsläufe gemittelt, in den weiteren zwei Fällen über jeweils zwei Durchläufe.

Die Diagramme zeigen die Abweichung des besten Individuums vom Optimum (937) über der Anzahl an Generation aufgetragen.

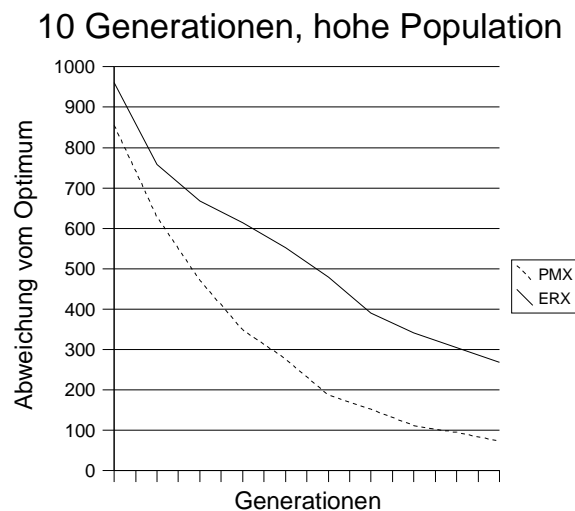
Bei einer Populationsgröße von 500 zeigt sich die schnellere Konvergenz des PMX Operators. Die Ergebnisse sind allerdings noch relativ weit vom Optimum entfernt.



Folgende Abbildung zeigt, dass der ERX Operator auf Dauer bessere Ergebnisse liefert, Populationsgröße war 300.



Der PMX Operator profitiert weit mehr von größeren Populationsgrößen (hier 10 000) bei geringer Generationenanzahl (hier 10) als der ERX Operator. Größere Generationenanzahlen habe ich nicht getestet, da die Laufzeit erheblich ist. Auch ersichtlich wird, dass hier die besten Touren der Ausgangspopulation nicht wesentlich besser sind als bei kleiner Population.



7. Quellen

- [1] Zbigniew Michalewicz: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag 1992
- [2] E. Schöneburg, F. Heinzmann, S. Fodderon: Genetische Algorithmen und Evolutionsstrategien, Addison-Wesley 1996

Verschiedene TSPs, teilweise mit Lösung:

- [3] <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>